CALIFORNIA INSTITUTE OF TECHNOLOGY

Computer Science Department

Silicon Structures Project

TECHNICAL REPORT #3230

Models for Structured IC Design

by

J.P. Gray

and

I. Buchanan

November, 1979

Models for Structured Integrated Circuit Design

T. Buchanan

Department of Computer Science

University of Edinburgh

and

J.P. Gray

Department of Computer Science

California Institute of Technology

ABSTRACT

Traditional design tools based on geometric representations do not provide an adequate base from which to construct and verify silicon implementations of complex systems. More comprehensive structural, physical and behavioral descriptions must be developed from appropriate representations. This paper proposes models which may be used to construct unified and consistent descriptions of the structural, physical and behavioral attributes of a design and also discusses a method of capturing these descriptions using a textual representation embedded in an object oriented programming language.

CONTENTS

1.0 Introduction

The development of integrated circuit fabrication techniques has led to an exponential increase in the number of transistors that it is possible to put on one chip. It is already possible to design single chip implementations of systems comprising over 100000 transistors and an order of magnitude increase in density over the next few years is probable [1]. Without adequate design tools designers will find it impossible to develop, partition and check very large scale integrated (VLSI) circuit designs.

Large designs cannot be addressed simply by stretching existing design aids. A structured design style[2] is essential. Within such a context, an accurate and comprehensive model of the design must be constructed at each level of abstraction. At the lowest level a circuit consists of primitive components ie wires and transistors. A collection of connected components forms a module called a block definition. Block instances may be included within other block definitions to build hierarchical design descriptions.

To connect components together in a way which preserves their structural, physical and behavioral significance requires the introduction of a new object which may be referenced as the node (structural data), or connect point (physical data) of a number of components. As this object has both physical and structural attributes it has been named a coordinode. Its attributes are discussed in Section 3.1.

This paper also describes the use of an object oriented language, SIMULA, used to model coordinodes, primitive components, and block structure. Data and procedural attributes may be associated with the system defined or user constructed objects. Within the paper a discussion of the model in general terms precedes a more detailed account of the SIMULA implementation.

2.0 A Structured Design Style

The principles of structured design already familiar from other disciplines eg computer programming, can also be applied to integrated circuit design [2]. Modularity, hierarchy, locality and regularity are always important considerations in any design activity which requires control of a high level of complexity. VLSI design falls into the category of high complexity design problems.

In general, a VLSI design can be separated logically into descriptions at a number of levels of abstraction. These may be formalized eg PMS and RT levels, or may be levels which are not formally recognised but, within a

hierarchical design, represent progressively smaller module grain sizes. The number of levels will increase with the complexity of the design. At any level a block may be defined which consists of a collection of connected components. Components may be system provided primitives ie wires and transistors, or may be instances of block definitions held in a library or defined by the designer. The complete design is contained in the highest level block definition.

It is assumed that there is a simple mapping between block instances in a structural definition and areas of silicon in a physical layout. This assumption is based on the notion that the structural partition of a complex design is developed from a primary concern for wiring management. Thus from every hierarchical structural specification there is a similar physical design specification of nested cells. A paraphrase of this constraint is that, for a given block definition, there is a simple transformation from a structural block diagram to a cell floor plan. This unification of the structural and physical hierarchies should provide a simplification in CAD system design provided the appropriate representations can be devised to capture both descriptions simultaneously.

Note that blocks have structural, physical and behavioral properties which a design system must capture in their entirety in order to construct a complete and consistent description of the design to use in verification and fabrication. Various graphical and textual notations, both formal and informal, can be employed in design description. This paper chooses to concentrate on a textual design representation expressed in an object oriented programming language, SIMULA. Such a language provides the necessary block level constructs within which the object oriented features of the language match the constructive and manipulative features of the computer aided design activity.

3.0 Models for Integrated Circuit Design

Models are used to build design descriptions that may be algorithmically tested. The testing, or verification, processes include dimensional design rule checking, wiring audit, loading analysis, functional simulation ,timing ,simulation,etc. Different data are required for each process and this data is usually extracted from two of the three fundamental descriptions of the design. For example, simulation uses structural and behavioral data together with test stimulii to produce circuit responses. Also, for pathology-free dimensional design rule checking it is necessary to know something of the structural definition of a design. Thus, in general, verification requires that the structural, physical and behavioral attributes of the design be known to the design system. In addition, these

descriptions must be complete and consistent at every hierarchical level of the design from primitive components up to high level blocks to achieve reliable verification. Higher level behavioral descriptions can be captured using procedural models in the normal way. The most important need, therefore, is for models to describe a design that allows the capture of structural and physical data. Proposals for a sufficient set of models are given in the next section. These are based on notions of capturing design intent by using initial topological specifications.

3.1 The Coordinode

The coordinode is a named object which has structural, physical and behavioral significance in the IC design process.

Structurally its function is to join together components ie wires, transistors and block instances. From a topological standpoint, a graph model could be constructed in which coordinodes are the nodes while components are the branches. A set of components which reference the same coordinode are connected by it. Figure 1 shows a stick diagram representation of a block containing half a shift register cell with the coordinodes identified by name. An associated physical layout appears below the stick diagram.

In addition, a coordinode may be designated a "pin" ie an interface point to an external environment, or a "contact" ie a reference to components on more than one layer that the designer intends to be joined. The absence of a contact under such circumstances is held to be an error since design intent must be positively reinforced. More than one coordinode may exist at the same physical position while having different structural characteristics.

Physically, the coordinode is placed in the 2D plane. This fixes the position and extent of primitive components since they are sited relative to coordinodes. A coordinode which acts as a contact has a physical realization which is the appropriate geometry for the type of contact. The designer may define the orientation of a butting contact. Pins have no geometric significance.

Behaviorally, the coordinode belongs to an electrical node or net which carries a simulation state between the components it joins.

To summarize, the attributes and functions of a coordinode are:

i) To provide connections between components.

ii) To provide inter-layer connections by the contact

mechanism.

iii) To provide block instance connections by the pin mechanism.

iv) To place components within blocks and define their sizes.

v) To pass simulation states between components, block instances and procedural definitions of block behavior.

3.2 Primitive Components

Traditional IC design concerned itself with only the geometric, usually polygonal, description of the circuit. While this representation is adequate for fabrication it is not suitable for a generalized structural description and associated verification procedures. In any CAD system it is necessary to define a number of primitive components for modelling purposes. The art is to choose models that allow structural, physical and behavioral design intent to be captured concurrently while maintaining consistency of the design over these representations at all times.

3.2.1. The Wire

The simplest component is a wire. It exists on a single layer and connects two coordinode endpoints by a path whose width defaults to a layer associated minimum (Figure 2).

Structurally, or topologically, the wire is a single branch in the circuit graph. Connected wires form a net ie an electrical node in the circuit.

Geometrically, the wire is a polygon constructed by inflating its associated path by half its specified width. This causes extensions to be formed at the wire endpoints and ensures valid physical connection between wires and other components. Some implementations may find the production of circular arcs an attractive option in the inflation process [3].

Behaviorally, a wire is part of an electrical node which may connect several components.

3.2.2 The Transistor

The transistor is the key primitive functional element (Figure 3). Different types of transistor may be modelled in the system eg pullups, pulldowns and pass transistors. Transistors are given textual names by the designer.

Structurally, the transistor has gate (input and output), source and drain connections.

Physically, the transistor possesses a path either between its gate input and output or its source and drain. The default path is determined by the type of transistor. This defines the electrical parameters of the device and its geometry. Note that the physical realization of the transistor extends outside the overlap area to include the abutting design rule enforced regions on the polysilicon and diffusion layers. Additionally, depletion mode transistors have an associated implant geometry.

Behaviorally, the different types of transistor act as specified by their associated models and according to the simulation states present at the coordinode connection points.

Pass transistors require special attention as they are bilateral devices.

4.0 Block Structure

A structured IC design methodology derives its power from the control of the circuit complexity that may be gained by partitioning the circuit into hierarchical blocks.

4.1 The Block Definition

Blocks are disjoint modules with structural, physical and behavioral descriptions. Structurally they consist of a set of components referencing a set of coordinodes. The components may be primitives or instances of other blocks.

Physically a block is a bounded region on the surface of the silicon chip. Both bounding boxes and bounding polygons may be used in the physical description but the box alternative is more efficient for cell arrays, chip assembly and computation while not being unduly restrictive within a structured design methodology. Behaviorally, a block performs as the aggregate of its parts, although when instanced it may have an associated procedural model that has been provided by the user.

The regularity of the design ie the use of arrays of cells and the control of interconnect, further contributes to the reduction in the volume of data. With this goal in mind a single block may be instanced many times. A block instance may be referenced within another block definition so that a hierarchical structure with a number of levels of block instancing can be built.

4.2 Block Parameterization

A block definition may be parameterized to achieve a number of effects important in chip assembly [4].

i) deformation or stretchability

ii) variable dimensions in arrays

iii) programmability of function eg ROM, PLA

iv) conditional inclusion of circuit elements

Thus parameterization is an extremely powerful tool in the construction of a design enabling large savings of effort in designing separate cells for similar functions. For example, deformations and iterations of a half shift register cell as shown in Figure 4.

4.3 The Block Instance

The block instance is a named object which may be viewed as a user defined component. It references its parent definition and is parameterized by a transformation matrix.

Structurally, a block instance is a set of pins interfacing to other components contained in a block definition. Thus pins specified in the block definition are translated into coordinodes in the calling environment within which they have the appropriate structural significance.

Physically, a block instance is fully expanded into its components for fabrication but may be represented as an outline on check plots and in dimensional design rule checking.

The behavioral characteristics of a block instance may be inferred from its parent definition either by expansion of the contents of the definition or by execution of a simulation procedure. In either case the block instance must preserve its local state.

5.0 SIMULA Implementation

An IC design may be described within a special purpose SIMULA programming environment. This environment provides a block structuring mechanism within which cell definitions may be constructed and includes procedures for generating and connecting components. The normal CLASS parameterization can be applied to block definitions with great effectiveness for cell deformation and the SIMULA

sequence and control structures can be used for iterations and conditional composition.

A complete example of a shift register cell array is given in Appendix A.

5.1 The Block Definition Hierarchy

The special purpose programming environment provided by the system includes a CLASS "blockdef". Any user defined block definition is constructed as a subclass of blockdef eg

```
blockdef CLASS shiftregistercell;
```

This implies that the procedural attributes of blockdef are also in its subclasses. Specifically, procedures exist within "blockdef" which allow the designer to create primitive components and instances of other blocks within the new block definition.

Parameters to a block definition appear in the CLASS header eg

```
blockdef CLASS cell(gnd,vdd,signal);REAL gnd,vdd,signal;
                                    VALUE gnd,vdd,signal;
```

The designer includes a block instance in a block definition by calling a procedure which takes as parameters the name of the instance, a reference to a block definition object and a transformation matrix which physically positions the instance. Thus a "shiftregistercell" object may be created eg

```
REF (shiftregistercell) src;  src:-NEW shiftregistercell;
```

Within another block definition a procedure call may appear which references it, positioning the instance at the origin of the new definition. The orientation of the block instance is left unchanged by including the identity matrix as its transformation matrix eg

```
    blockinst("shiftcell",src,NEW transform(NONE));
```

Transformation matrices may be constructed to perform translation, rotation and reflection using procedures included in the CLASS transform.

A block instance is regarded as a user-defined component. Structurally it is viewed as a black box with a number of pins to its calling environment. Each pin is translated to a coordinode in the calling environment and is referenced by concatenating the instance name and pin name eg

"shiftcell.signalinput"

These names are used by the new block definition in attaching other components to the block instance.

Physically, the block instance is positioned by the transformation matrix and it follows that the placement of the coordinode in the calling environment is fixed by the same matrix. The block instance covers the same dimensions in physical area as its parent definition.

Behaviorally, the block instance may either act as the aggregate of the components contained in its parent definition or, if the designer so chooses, perform according to a procedure included in the block definition.

Thus through successive block instancing at a number of levels a hierarchy of modules is constructed and the volume and complexity of design data is controlled.

5.2 Primitive Components

There are two basic primitive components: the wire and the transistor. Structurally, wires connect two coordinodes and exist on a single layer eg

```
wire("from","to",poly);
```

Physically, a wire is a path joining two points in the 2D plane. The default path is a straight line. Any other path is constructed by system provided procedures which describe the increments for each jog of the path [5]. Both absolute and relative increments are permitted and jogs may be paraxial or generalized vectors eg

```
wire("from","to",poly).path.x(10).dy(1).xy(12,6);
```

The specification of L-shaped jogs may be abbreviated using the procedures "xtheny" and "ythenx" eg

```
wire ("from","to",poly).path.xtheny;
```

Width is a layer associated default unless changed explicitly by the designer eg

```
wire("from","to",poly).width(10);
```

Behaviorally, a wire is one component in an electrical node or net.

There are a number of different types of transistors. Pullups, pulldown and pass transistors are currently defined in the system. The models for each type differ in some respects but are basically the same.

Structurally, the transistor is a black box with four connections. These connections are pre-named "src","drn","in" and "out". A procedure is provided by the system which includes a particular type of transistor within a block definition and allows the designer to name it eg

```
pulldown("inv");
```

This statement causes coordinodes, named by concatenating the transistor name and connector names, to be entered into the block definition where they may be referenced by other components eg

```
"inv.src"
```

Physically, the transistor is a diffusion and polysilicon overlap area, whose dimensions are determined by the placement of its coordinode connections. Design rules enforce abutting extension regions of particular widths in the polysilicon and diffusion layers. Pullups are plotted and fabricated with an implant layer. Pulldowns are assumed to need a gate extension region, which by default continues in the same direction as the gate path, but may be specified by the designer. The gate path itself follows the same defaults as wire paths with the same arrangement for defaults.

Behaviorally, each transistor acts according to its particular system defined model.

5.3 Coordinode Attributes and Placement

Coordinodes may be explicitly assigned one of two attributes, pin and contact.

```
pin("sigin");  contact("gndx");
```

A pin is simply a structural attribute which defines a block's interface to its calling environment.

A contact acts as confirmation by the designer that components on different layers which reference the same coordinode are logically connected. Physically, contacts require a contact hole and a design rule enforced overlap on the connecting layers. Butting contacts are placed by the system if a polysilicon to diffusion connection is required. The orientation of the contact may be deduced from the directions of its connections in the direction of the diffusion overlap but it may need to be explicitly defined by a vector.

```
contact("buttx").orientation (1,0);
```

Coordinodes are placed at a point in the 2D plane, which may

be a function of the block definition parameters. For example, baselines are positions on the x or y axis which may be parameters to the block definition and may be used to produce stretchable cells [4]. Coordinodes may also be placed relative to another's base.

```
     place("sigin",NEW point(0,sig));

     place("inv.src",NEW point(inv+3,sig+4));

place("inv.drn",position("inv.src").plus (NEW point(0,2)));
```

6.0 Conclusions

By unifying the structural, physical and behavioral representations of an IC design, the system described in this paper provides the designer with a tool which can cope with the complexity of VLSI designs in a structured environment. The current system can, or soon will be able to, produce graphical and fabrication oriented output, perform dimensional and electrical design rule checks, simulate the design at the logic circuit level and construct a wiring list suitable for input to a circuit simulation program.

A special purpose language would improve the usability of the system by decreasing the verbosity of the descriptions and an interactive graphics front end would enhance designer productivity.

References:

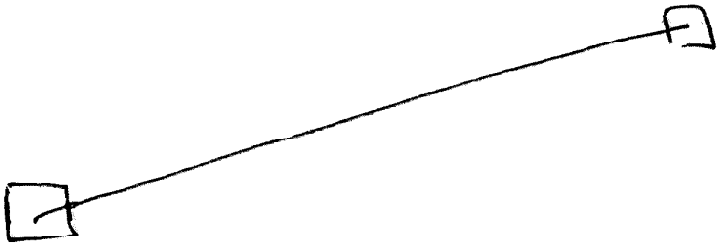
[1] I.E. Sutherland, C.A. Mead and T.E. Everhart "Basic Limitations in Microcircuit Fabrication Technology" Report R-1956-ARPA, RAND Corp., Santa Monica, Ca. 90406

[2] Mead and Conway "Introduction to VLSI Systems" Addison-Wesley, 1980.

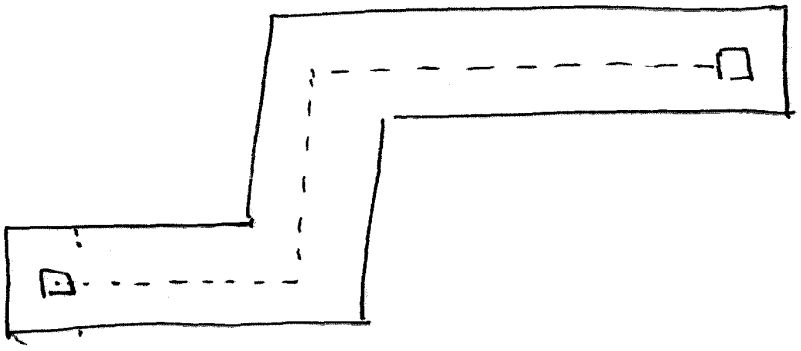
[3] E. Barton and I. Buchanan "The Polygon Package" CAD Journal, Jan 1980.

[4] D. Johannsen "Bristle Blocks: A Silicon Compiler" Proceedings 16th Design Automation Conference, June 1979.

[5] R. Locanthi "LAP: A SIMULA Package for IC layout" Caltech Display File 1862, July 1978.

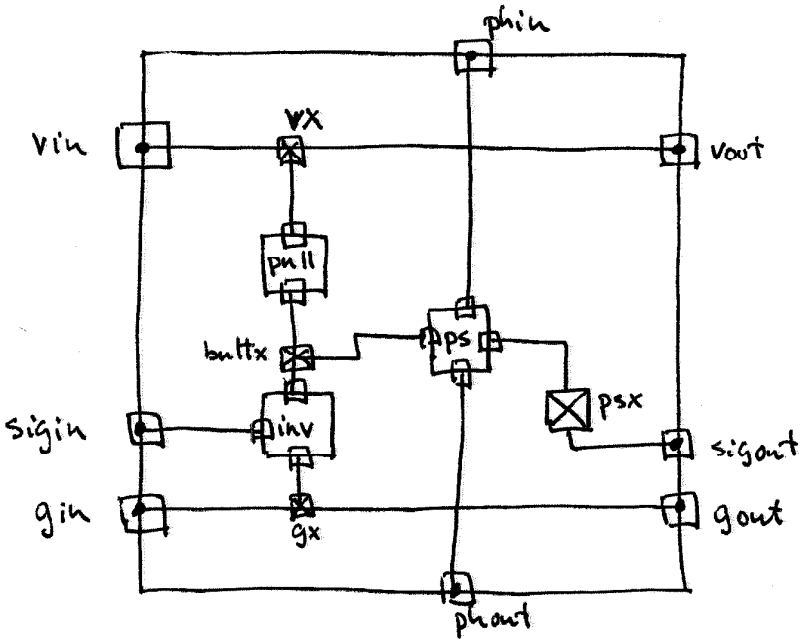


(a) Structural

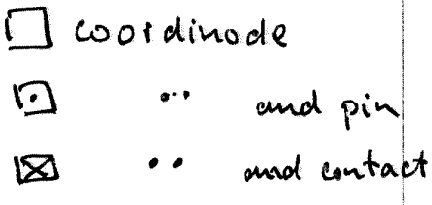


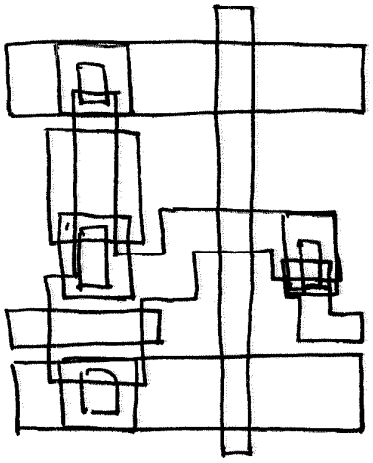
(b) Physical.

Figure 2. Wires



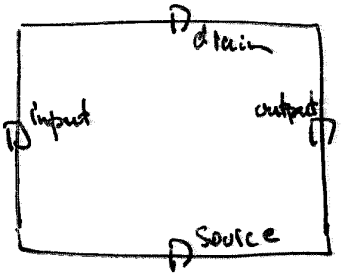
(a) Structural Definition



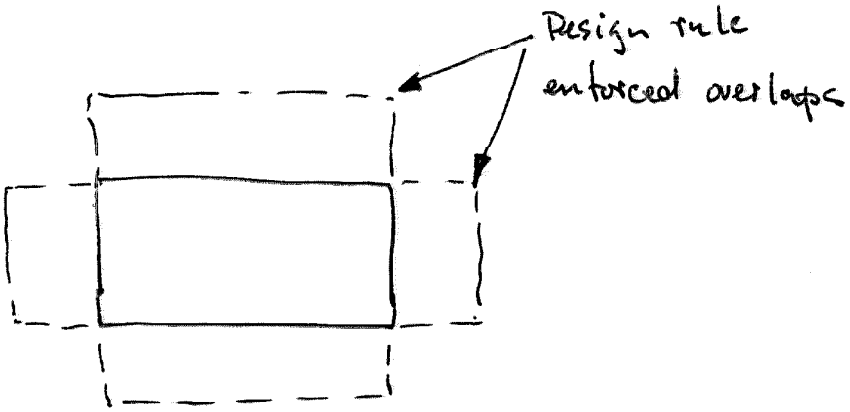


(b) Physical Definition

Figure 1. Coordinodes



(a) Structural.



(b) Physical.

Figure 3 Transistors.

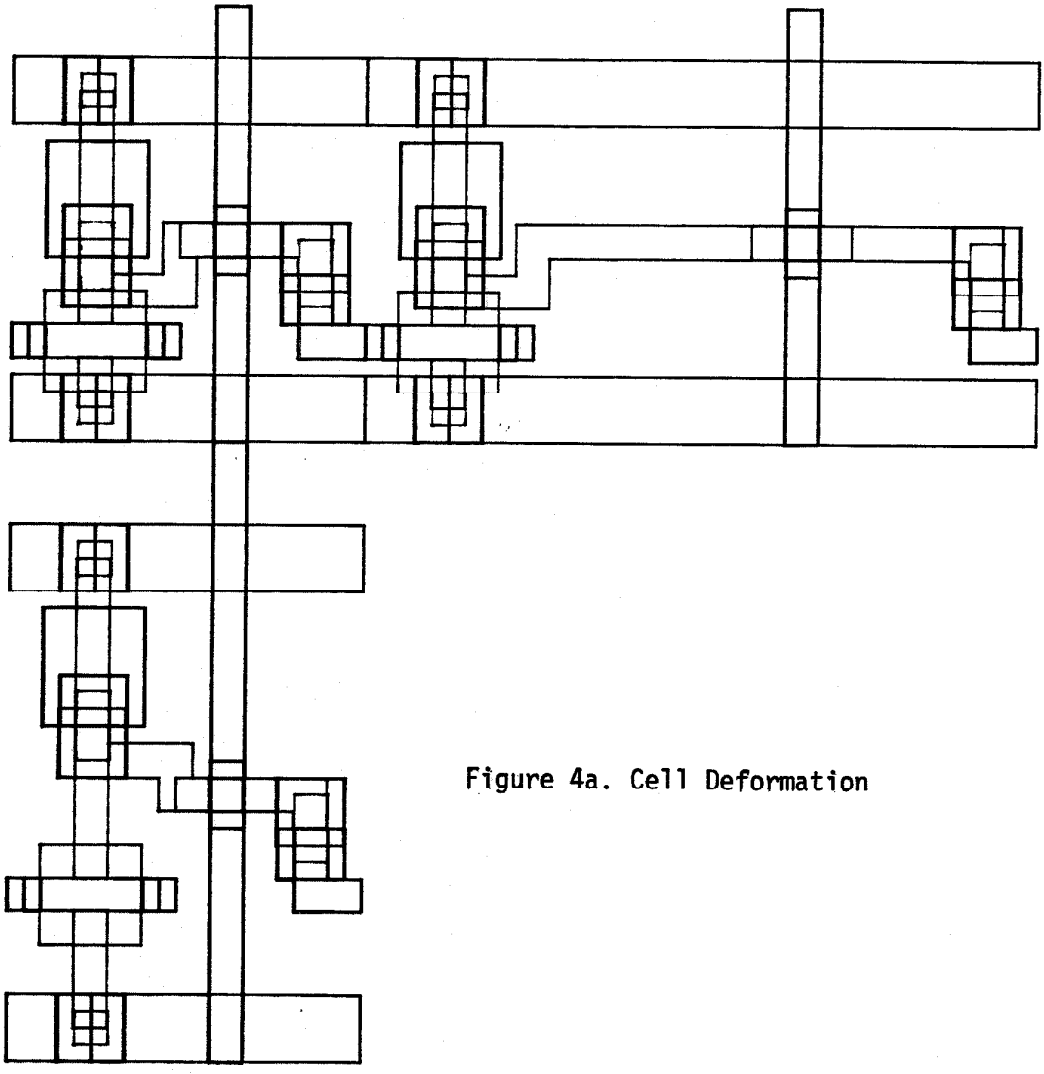


Figure 4a. Cell Deformation

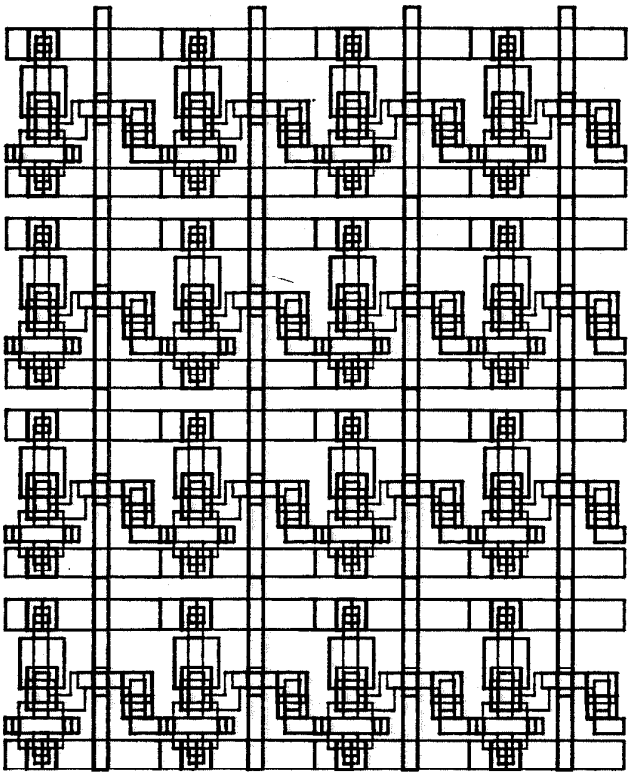


Figure 4b. Cell Iteration

Appendix 1. SIMULA Cell Descriptions

```
blockdef CLASS srcelldef (gnd, sig, vdd, phi, xlim, ylim,type);
VALUE type; TEXT type;   REAL gnd,sig, vdd, phi, xlim, ylim;

BEGIN
    INTEGER invbase,ypass;


    !set text name of cell definition;
    nameblock (conc ("srcell.",type));

    ! check default settings for parameters;
    default (gnd,2); default (sig,6); default (vdd,21);
    default (phi,13); default (xlim,21); default (ylim,26);
    !define block bounding box;
    blockboundbox (0,0,xlim, ylim);

    !define the inverter x baseline and;
    !the pass transistor y baseline;
    invbase:= 5; ypass:=sig+6;

    !external interface;
    pin("gin");pin("gout");pin("vin");pin("vout");
    pin("phin");pin("phout");pin("sigin");pin("sigout");

    !main components;
    passtran("ps");pulldown("inv");pullup("pull","buttx");

    !wires;
    !power and ground lines;
    wire("gin","gx",metal).width (4); wire("gx","gout",metal).width (4);
    wire("vin","vx",metal).width (4); wire("vx","vout",metal).width (4);

    !clock line;
    wire("phin","ps.in",poly);
    wire("ps.out","phout",poly);

    !pullup connections;
    wire("vx","pull.drn",diff);
    wire("pull.src","buttx",diff);

    !pulldown connections;
    wire("buttx","inv.drn",diff);
    wire("inv.src","gx",diff);

    !data line wires;
    wire("sigin","inv.in",poly);
    wire("buttx","ps.drn",diff).path.dy(-2).dx(5).ythenx;
    wire("ps.src","psx",diff).path.xtheny;
    wire("psx","sigout",poly).path.ythenx;

    !coordinodes with contact attributes;
    contact("gx");contact("vx");contact ("buttx").orientation (0,1);
    contact("psx").orientation (0,-1);

    !coordinode physical positions;

    !power and ground lines;
    place("gin",NEW point (0,gnd));
    place("gx",NEW point (invbase,gnd));
    place("gout",NEW point(xlim,gnd));
    place("vin",NEW point(0,vdd));
    place("vx",NEW point(invbase,vdd));
    place("vout",NEW point(xlim,vdd));
```

```
    !data and control pins;
    place("sigin",NEW point (0,sig));
    place("sigout",NEW point (xlim,sig));
    place("phin",NEW point (phi,ylim));
    place("phout",NEW point (phi,0));

    !pullup connections;
    place("pull.drn",NEW point (invbase,vdd-3));
    place("pull.src",NEW point (invbase,vdd-10));

    !inverter output to pass transistor;
    place("buttx",NEW point (invbase,vdd-10));

    !pulldown connections;
    place("inv.drn",NEW point (invbase,sig+1));
    place("inv.src",NEW point (invbase,sig-1));
    place("inv.in",NEW point (invbase-3,sig));
    place("inv.out",NEW point (invbase+3,sig));

    !pass transistor connection;
    place("ps.in",NEW point (phi,ypass+1));
    place("ps.out",NEW point (phi,ypass-1));
    place("ps.src",NEW point (phi+1,ypass));
    place("ps.drn",NEW point (phi-1,ypass));

    !signal connection output;
    place("psx",NEW point (xlim-3,sig+4));

END of srcelldef;



blockdef CLASS srcellarraydef (xrep,yrep,celldef,type);
VALUE xrep,yrep,type; TEXT type; REAL xrep,yrep; REF(srcelldef) celldef;
BEGIN
    INTEGER i,j,xcell,ycell,xrep2;

    !set text name of cell definition;
    nameblock(conc ("srcellarray.",type));

    !some useful numbers;
    xrep2:=xrep//2;
    xcell:=celldef.xlim;ycell:=celldef.ylim;

    !set block bounding box to enable;
    !coordinode position checks;
    blockboundbox (0,0,xcell*xrep,ycell*yrep);

    !pin specification;

    FOR i:=1 STEP 1 UNTIL xrep2 DO BEGIN
        pin(si("phi1in",i));pin(si("phi1out",i));
        pin(si("phi2in",i));pin(si("phi2out",i));
    END;

    FOR i:=1 STEP 1 UNTIL yrep DO BEGIN
        pin(si("gin",i));pin(si("gout",i));
        pin(si("sigin",i));pin(si("sigout",i));
        pin(si("vin",i));pin(si("vout",i));
    END;
```

```
    !shift register cell instance specification;

    FOR i:=1 STEP 1 UNTIL xrep DO BEGIN
        FOR j:=1 STEP 1 UNTIL yrep DO
        blockinst(s2("src",i,j),celldef,
        NEW transform (NONE).translatedby (NEW point
        ((i-1)*xcell,(j-1)*ycell)));
    END;

    !power, ground and data lines;

    FOR j:=1 STEP 1 UNTIL yrep DO BEGIN
        FOR i:=1 STEP 1 UNTIL xrep-1 DO BEGIN
            wire(conc(s2("src",i,j),".gout"),conc(s2("src",i+1,j),".gin"),metal);
            wire(conc(s2("src",i,j),".vout"),conc(s2("src",i+1,j),".vin"),metal);
            wire(conc(s2("src",i,j),".sigout"),conc(s2("src",i+1,j),".sigin"),poly);
        END;

    END;

    ! clock lines;

    FOR i:=1 STEP 1 UNTIL xrep DO BEGIN
        FOR j:=yrep STEP -1 UNTIL 2 DO
        wire(conc(s2("src",i,j),".phout"),conc(s2("src",i,j-1),".phin"),poly);

    END;

    !wires to pin;
    FOR i:=1 STEP 1 UNTIL yrep DO BEGIN
        wire(si("gin",i),conc(s2("src",1,i),".gin"),metal);
        wire(si("sigin",i),conc(s2("src",1,i),".sigin"),poly);
        wire(si("vin",i),conc(s2("src",1,i),".vin"),metal);
        wire(si("gout",i),conc(s2("src",xrep,i),".gout"),metal);
        wire(si("sigout",i),conc(s2("src",xrep,i),".sigout"),poly);
        wire(si("vout",i),conc(s2("src",xrep,i),".vout"),metal);

    END;

    FOR i:=1 STEP 1 UNTIL xrep2 DO BEGIN
        wire(si("phi1in",i),conc(s2("src",2*i-1,yrep),".phin"),poly);
        wire(si("phi1out",i),conc(s2("src",2*i-1,1),".phout"),poly);
        wire(si("phi2in",i),conc(s2("src",2*i,yrep),".phin"),poly);
        wire(si("phi2out",i),conc(s2("src",2*i,1),".phout"),poly);
    END;

    !place pins physically;

    FOR i:=1 STEP 1 UNTIL yrep DO BEGIN
        place(si("gin",i),position(conc(s2("src",1,i),".gin")));
        place(si("sigin",i),position(conc(s2("src",1,i),".sigin")));
        place(si("vin",i),position(conc(s2("src",1,i),".vin")));
        place(si("gout",i),position(conc(s2("src",xrep,i),".gout")));
        place(si("sigout",i),position(conc(s2("src",xrep,i),".sigout")));
        place(si("vout",i),position(conc(s2("src",xrep,i),".vout")));
    END;

    FOR i:=1 STEP 1 UNTIL xrep2 DO BEGIN
        place(si("phi1in",i),position(conc(s2("src",2*i-1,yrep),".phin")));
        place(si("phi1out",i),position(conc(s2("src",2*i-1,1),".phout")));
        place(si("phi2in",i),position(conc(s2("src",2*i,yrep),".phin")));
        place(si("phi2out",i),position(conc(s2("src",2*i,1),".phout")));
    END;

END of srcellarraydef;
```